CREDENTIAL LOADING AND THE AWS SDK FOR .NET (DEEP DIVE)

12th May 2020 (https://www.stevejgordon.co.uk/credential-loading-and-the-aws-sdk-for-dotnet-deep-dive) Steve Gordon (https://www.stevejgordon.co.uk/author/stevejgordon) .NET (https://www.stevejgordon.co.uk/category/net), AWS (https://www.stevejgordon.co.uk/category/aws)

In this post, I want to dive into how the AWS SDK for .NET attempts to load credentials for any service clients which you use in your applications. I'm going to focus specifically on .NET Core applications, where the SDK client(s) are resolved from the dependency injection (DI) container. However, the steps are relatively consistent, even if you are creating the client instances directly (not from the DI container) and even on .NET Framework.

TL;DR;

FROM WHERE AND IN WHAT ORDER DOES THE AWS .NET SDK LOAD CREDENTIALS?

1. From the **Credentials** property of AWSOptions, if an instance of AWSOptions is provided during registration.
2. **Shared Credentials File (Custom Location)**. When an instance of AWSOptions is provided,

and both the profile and profile location are specified.

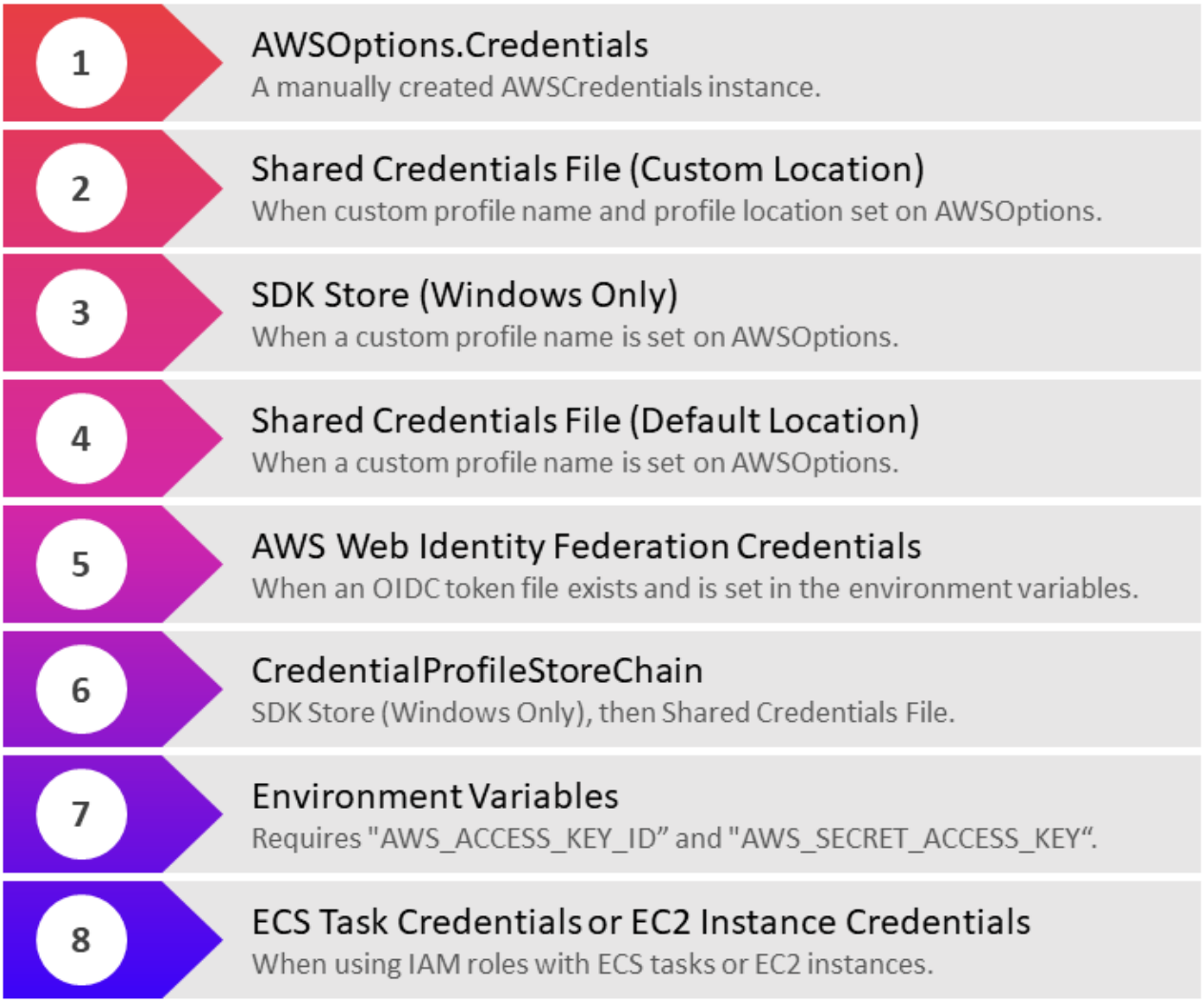
3. **SDK Store (Windows Only)**. When an instance of AWSOptions is provided and only the profile is set (profile location is null).
4. **Shared Credentials File (Default Location)**. When an instance of AWSOptions is provided and only the profile is set (profile location is null).
5. **AWS Web Identity Federation Credentials**. When an OIDC token file exists and is set in the environment variables.
6. **CredentialsProfileStoreChain**.
    1. SDK Store (if on Windows) encrypted using Windows Data Protection API.
    2. Shared Credentials File in the default location.
7. **Environment variables**. When the Access Key ID and Secret Access Key environment variables are set.
8. **ECS Task Credentials or EC2 Instance Credentials**. When using IAM roles with ECS tasks and ECS instances.

The above is correct as of version 3.3.101 of the AWS SDK for .NET. Merely knowing this order is usually enough when consuming the library in your applications. If you're like me though, you may be curious to understand a little more about the credential loading internals.

HOW AWS CREDENTIALS ARE LOADED

The AWSSDK.Extensions.NETCore.Setup NuGet package (https://www.nuget.org/packages/AWSSDK.Extensions.NETCore.Setup/) supports the integration of the AWS SDK for .NET with the .NET Core configuration and dependency injection frameworks. This package allows us to register the AWS service clients we need to use in our application so that they may be resolved from the dependency injection container. It's also possible to create an instance of the SDK service clients directly, in which case, many of the same steps for credential loading are also applied.

Internally, the extensions library uses a ClientFactory to create new instances of the service clients when they are required. This type, combined with some core internal mechanisms, will configure the service client instance by following several steps.



1: AWSOPTIONS.CREDENTIALS

When registering a service client with the DI container, we call the AddAWSService<T> extension method on the IServiceCollection. This method has a few overloads, one of which accepts an AWSOptions instance used to configure the service client.

```
1   services.AddAWSService<IAmazonSQS>(new AWSOptions
2   {
3       Credentials = new BasicAWSCredentials("123", "456")
4   });
```

view raw (https://gist.github.com/stevejgordon/e08493eb2cfe91f833a23f8b5a263e80/raw/c42b6e4ac2aa031b733bd1a385ae2a6dc09a8274/Startup.cs) Startup.cs (https://gist.github.com/stevejgordon/e08493eb2cfe91f833a23f8b5a263e80#file-startup-cs) hosted with ❤ by GitHub (https://github.com)

In the preceding example, we set the Credentials on the AWSOptions using an instance of BasicAWSCredentials. This is really bad idea since we're directly exposing our credentials in the source code and likely our source control history. Don't use it in this way for real applications!

AddAWSService adds a ServiceDescriptor to the IServiceCollection as follows:

```
1   public static IServiceCollection AddAWSService<T>(this IServiceCollection collection, AWSO
2   {
3       Func<IServiceProvider, object> factory =
4           new ClientFactory(typeof(T), options).CreateServiceClient;
5
6       var descriptor = new ServiceDescriptor(typeof(T), factory, lifetime);
7       collection.Add(descriptor);
8       return collection;
9   }
```

view raw (https://gist.github.com/stevejgordon/d98b47f42300374d45e6150c3f861983/raw/fa7edecdd3e5b0bfb356a02d0ae71b6255fe3277/ServiceCollectionExtensions.cs) ServiceCollectionExtensions.cs (https://gist.github.com/stevejgordon/d98b47f42300374d45e6150c3f861983#file-servicecollectionextensions-cs) hosted with ❤ by **GitHub (https://github.com)**

This method provides a factory registration, Func<IServiceProvider, object>, which gets called anytime a new instance of the service client is required. Note that by default, service clients are registered with the singleton lifetime, so only a single instance is created when it is first needed to fulfil a dependency. The Func registered here, creates a new ClientFactory which accepts an AWSOptions parameter. AWSOptions may be null or may be an instance provided in the registration as I showed above. The service client is then created by calling the CreateServiceClient method. Don't worry about this detail too much for this post; we'll focus on what comes next.

When creating the service client, the first step the ClientFactory completes is to load the AWS credentials, which it will provide to the service client. This takes place inside a CreateCredentials method.

```
1   private static AWSCredentials CreateCredentials(ILogger logger, AWSOptions options)
2   {
3       if (options != null)
4       {
5           if (options.Credentials != null)
6           {
7               logger?.LogInformation("Using AWS credentials specified with the AWSOptions.C
8               return options.Credentials;
9           }
10          if (!string.IsNullOrEmpty(options.Profile))
11          {
12              var chain = new CredentialProfileStoreChain(options.ProfilesLocation);
13              AWSCredentials result;
14              if (chain.TryGetAWSCredentials(options.Profile, out result))
15              {
```

```
16                  logger?.LogInformation($"Found AWS credentials for the profile {options.P
17                  return result;
18              }
19              else
20              {
21                  logger?.LogInformation($"Failed to find AWS credentials for the profile {
22              }
23          }
24      }
25
26      var credentials = FallbackCredentialsFactory.GetCredentials();
27      if (credentials == null)
28      {
29          logger?.LogError("Last effort to find AWS Credentials with AWS SDK's default cred
30          throw new AmazonClientException("Failed to find AWS Credentials for constructing
31      }
32      else
33      {
34          logger?.LogInformation("Found credentials using the AWS SDK's default credential
35      }
36
37      return credentials;
38  }
```

view raw (https://gist.github.com/stevejgordon/5fb5de80050f096ad518ce8a0330c31e/raw/a94830b77e04431eaf930da47273ed2cd6d006ef/ClientFactory.cs) ClientFactory.cs (https://gist.github.com/stevejgordon/5fb5de80050f096ad518ce8a0330c31e#file-clientfactory-cs) hosted with ❤ by GitHub (https://github.com)

If an AWSOptions instance were provided when registering the AWS service client, it would be not null at the point that this method is called. The Credentials property on the AWSOptions class may hold a reference to a manually created AWSCredentials instance, which will be used by the service client if it is available. This is, therefore, the first choice for credentials which may be applied to the service client.

2: SHARED CREDENTIALS FILE (CUSTOM PROFILE NAME AND LOCATION)

The next conditional occurs if the Profile property on the AWSOptions has a value. This is expected to be the name of a profile from which to load the credentials. AWS supports declaring multiple named profiles in some of the possible credential files.

We could, for example, register our service with AWSOptions specifying the use of a profile named custom.

```
services.AddAWSService<IAmazonSQS>(new AWSOptions
{
    Profile = "custom",
    ProfilesLocation = @"c:\temp\credentials"
});
```

view raw (https://gist.github.com/stevejgordon/48714820168ed7e078d8e24beb55c6db/raw/f3e508ca07edeaab1ff42222228a4996cf75450f/Startup.cs) Startup.cs (https://gist.github.com/stevejgordon/48714820168ed7e078d8e24beb55c6db#file-startup-cs) hosted with ❤ by GitHub (https://github.com)

In this scenario, an instance of the CredentialProfileStoreChain class is created within the CreateCredentials method on the ClientFactory. As a reminder, here's the relevant code again.

```
var chain = new CredentialProfileStoreChain(options.ProfilesLocation);
AWSCredentials result;
if (chain.TryGetAWSCredentials(options.Profile, out result))
{
    logger?.LogInformation($"Found AWS credentials for the profile {options.Profile}");
    return result;
}
else
{
    logger?.LogInformation($"Failed to find AWS credentials for the profile {options.Prof
}
```

view raw (https://gist.github.com/stevejgordon/e8aaf706a5382dff99a5e9275f58f986/raw/3e0987763420b4297a3cc65689c85ee93f256e36/ClientFactory.cs) ClientFactory.cs (https://gist.github.com/stevejgordon/e8aaf706a5382dff99a5e9275f58f986#file-clientfactory-cs) hosted with ❤ by GitHub (https://github.com)

The CredentialProfileStoreChain is created passing in the ProfilesLocation (which may be null) from the AWSOptions. The TryGetAWSCredentials method is called passing in the specified profile name. This, in turn, calls down to a method named TryGetProfile.

```
public bool TryGetProfile(string profileName, out CredentialProfile profile)
{
    if (string.IsNullOrEmpty(ProfilesLocation) && UserCrypto.IsUserCryptAvailable)
    {
        var netCredentialsFile = new NetSDKCredentialsFile();
        if (netCredentialsFile.TryGetProfile(profileName, out profile))
        {
            return true;
        }
    }

    var sharedCredentialsFile = new SharedCredentialsFile(ProfilesLocation);
```

```
13        if (sharedCredentialsFile.TryGetProfile(profileName, out profile))
14        {
15            return true;
16        }
17
18        profile = null;
19        return false;
20    }
```

view raw (https://gist.github.com/stevejgordon/c8ae078a03162dcc76791eb72305dcec/raw/eb6f667fb043d1d28d364992a6763a2059974552/CredentialProfileStoreChain.cs) CredentialProfileStoreChain.cs (https://gist.github.com/stevejgordon/c8ae078a03162dcc76791eb72305dcec#file-credentialprofilestorechain-cs) hosted with ❤ by GitHub (https://github.com)

When the ProfilesLocation is not null, then this will used to try to access a shared credentials file at that location. The shared credentials file stores credentials in plain text and can be accessed by various AWS tools such as any of the AWS SDKs, the AWS CLI and AWS Tools for PowerShell. It includes credentials for one or more profiles.

```
1   [default]
2   aws_access_key_id = 1111
3   aws_secret_access_key = 2222
4
5   [custom]
6   aws_access_key_id = 1234
7   aws_secret_access_key = 9999
```

view raw (https://gist.github.com/stevejgordon/0b088f1e743121eb7de5df08c499366d/raw/c66a50124745bb3b413181839ba79b5bd67e1197/credentials) credentials (https://gist.github.com/stevejgordon/0b088f1e743121eb7de5df08c499366d#file-credentials) hosted with ❤ by GitHub (https://github.com)

The credentials file from the supplied profiles location will be loaded and searched for a profile matching the Profile property from the AWSOptions. It's possible that a matching section for the profile will not be found in the shared credentials file.

3: SDK STORE (.NET SDK CREDENTIALS FILE) – WINDOWS ONLY (CUSTOM PROFILE NAME)

When the TryGetProfile method (above) is called on a CredentialProfileStoreChain that was created with a null profile location, its preference, when the platform supports it, is to attempt to load credentials from the .NET SDK credentials file (SDK Store). Credentials in the SDK Store are encrypted and reside in the current user's home directory. This helps to limit the risk of accidental exposure of the credentials. This functionality depends on the Windows Crypt32.dll module being available. Credentials contained within the SDK Store can be used by the AWS SDK for .NET, AWS Tools for Windows PowerShell and the AWS Toolkit for Visual Studio.

If cryptography is available (on Windows), a new instance of NetSDKCredentialsFile is created. This supports loading credentials which have been stored under the current users AppData folder encrypted using Windows Data Protection API. A profile with a matching name (or default) will be located if it exists in the store and be returned. The SDK Store is located in the C:\Users \<username>\AppData\Local\AWSToolkit folder in the RegisteredAccounts.json file.

4: SHARED CREDENTIALS FILE (CUSTOM PROFILE NAME AND DEFAULT LOCATION)

In cases, where the ProfilesLocation is null, and the platform does not support the SDK Store, then the shared credentials file in the default location will be searched for a matching profile. The default location for the credentials file is within a directory named ".aws" in the home directory of the current user. For example:

C:\Users\stevejgordon\.aws\credentials

When present, the file from this default location will be loaded and parsed to see if it contains a matching profile name. If the profile is located, the SDK attempts to create the credentials instance from the loaded values.

FALLBACKCREDENTIALSFACTORY

If no profile name was supplied to the CreateCredentials method, then the process continues and uses a class named FallbackCredentialsFactory to attempt to load credentials from several fall back options.

FallbackCredentialsFactory is a static class, which includes a static ctor which calls a static Reset() method.

Here is some of the relevant code inside FallbackCredentialsFactory.cs

```
1   public delegate AWSCredentials CredentialsGenerator();
2   public static List<CredentialsGenerator> CredentialsGenerators { get; set; }
3
4   public static void Reset()
5   {
6       Reset(null);
7   }
8
9   public static void Reset(IWebProxy proxy)
10  {
11      cachedCredentials = null;
12      CredentialsGenerators = new List<CredentialsGenerator>
13      {
14          () => AssumeRoleWithWebIdentityCredentials.FromEnvironmentVariables(),
```

```
15          // Attempt to load the default profile.  It could be Basic, Session, AssumeRole, 
16          () => GetAWSCredentials(credentialProfileChain),
17          () => new EnvironmentVariablesAWSCredentials(), // Look for credentials set in en
18          () => ECSEC2CredentialsWrapper(proxy),      // either get ECS credentials or inst
19      };
20  }
```

view raw (https://gist.github.com/stevejgordon/5c6041a55c7730f8a67188a4a135d622/raw/fbfd2c35db4087b6f60fa14773291280b07996bf/FallbackCredentialsFactory.cs) FallbackCredentialsFactory.cs (https://gist.github.com/stevejgordon/5c6041a55c7730f8a67188a4a135d622#file-fallbackcredentialsfactory-cs) hosted with ❤ by GitHub (https://github.com)

During reset, any cached credentials are cleared.

FallbackCredentialsFactory includes a delegate member “CredentialsGenerator” which defines a method which accepts no arguments and returns an instance of AWSCredentials. A list of these delegates is populated by the reset method.

In the NetStandard case (which we’ll focus on here) four delegates are added to the list in a specific order (we’ll come to that very shortly). After creating an instance of the FallbackCredentialsFactory, the ClientFactory.CreateCredentials code, calls its GetCredentials method. Its code is as follows.

```
1   public static AWSCredentials GetCredentials(bool fallbackToAnonymous)
2   {
3       if (cachedCredentials != null)
4           return cachedCredentials;
5
6       List<Exception> errors = new List<Exception>();
7
8       foreach (CredentialsGenerator generator in CredentialsGenerators)
9       {
10          try
11          {
12              cachedCredentials = generator();
13          }
14          catch (ProcessAWSCredentialException)
15          {
16              throw;
17          }
18          catch (Exception e)
19          {
20              cachedCredentials = null;
21              errors.Add(e);
22          }
```

```
23
24          if (cachedCredentials != null)
25              break;
26      }
27
28      if (cachedCredentials == null)
29      {
30          if (fallbackToAnonymous)
31          {
32              return new AnonymousAWSCredentials();
33          }
34
35          using (StringWriter writer = new StringWriter(CultureInfo.InvariantCulture))
36          {
37              writer.WriteLine("Unable to find credentials");
38              writer.WriteLine();
39              for (int i = 0; i < errors.Count; i++)
40              {
41                  Exception e = errors[i];
42                  writer.WriteLine("Exception {0} of {1}:", i + 1, errors.Count);
43                  writer.WriteLine(e.ToString());
44                  writer.WriteLine();
45              }
46
47              throw new AmazonServiceException(writer.ToString());
48          }
49      }
50
51      return cachedCredentials;
52  }
```

view raw (https://gist.github.com/stevejgordon/c110174cbc91f5093defa1adb4804fab/raw/c9368f60deef6c877cfc7082ebfb38d6409ad89f/FallbackCredentialsFactory.cs) FallbackCredentialsFactory.cs (https://gist.github.com/stevejgordon/c110174cbc91f5093defa1adb4804fab#file-fallbackcredentialsfactory-cs) hosted with ❤ by **GitHub (https://github.com)**

This code loops over each registered CredentialsGenerator delegate and invokes it. The delegates will either return an instance of AWSCredentials or will throw an exception. If and when one of the generators successfully provides the AWSCredentials, the instance is cached (stored in the cachedCredentials field) and the foreach loop breaks, returning the credentials.

5: AWS WEB IDENTITY FEDERATION CREDENTIALS

AWS, it's possible to allow login through an OpenID Connect (OIDC)-compatible identity provider. In such cases, you will be issued a token by the OIDC IdP which is expected to be stored

in a file.

The first delegate, which is added, calls the AssumeRoleWithWebIdentityCredentials.FromEnvironmentVariables method. This expects to load values from the environment variables which define the user of an OIDC provider for temporary, token-based access by assuming a role.

When any of the required environment variables are missing, an exception will is thrown, most probably an ArgumentNullException because the "AWS_WEB_IDENTITY_TOKEN_FILE" variable will not contain a value. Should all valid values be in place, an instance of AssumeRoleWithWebIdentityCredentials will be properly constructed. This provides refreshing credentials which will be refreshed every 5 minutes.

6: CREDENTIALPROFILESTORECHAIN

The second delegate in the list will attempt to load a profile using the CredentialProfileStoreChain. The registered delegate calls into the FallbackCredentialsFactory.GetAWSCredentials method, passing in a CredentialProfileStoreChain. A shared, static instance of the CredentialProfileStoreChain is stored in a private field of the FallbackCredentialsFactory. You'll recall that we saw the CredentialProfileStoreChain used earlier as well. In that case, it was only called if a custom profile name had been provided on the AWSOptions. At this stage, the profile name will either by the value of the "AWS_PROFILE" environment variable, if present, or will be "default".

```
1   private static AWSCredentials GetAWSCredentials(ICredentialProfileSource source)
2   {
3       var profileName = Environment.GetEnvironmentVariable(AWS_PROFILE_ENVIRONMENT_VARIABLE)
4
5       CredentialProfile profile;
6       if (source.TryGetProfile(profileName, out profile))
7           return profile.GetAWSCredentials(source, true);
8       throw new AmazonClientException("Unable to find the '" + profileName + "' profile in C
9   }
```

view raw (https://gist.github.com/stevejgordon/9ed5e68f6c2151f626d99c891e193636/raw/8115f834ae6ff8092d522a27f19d64814bc57cab/FallbackCredentialsFactory.cs) FallbackCredentialsFactory.cs (https://gist.github.com/stevejgordon/9ed5e68f6c2151f626d99c891e193636#file-fallbackcredentialsfactory-cs) hosted with ❤ by **GitHub (https://github.com)**

GetAWSCredentials will attempt to load credentials from various sources by providing the profile name. The code will then try to load a profile from the CredentialProfileStoreChain. On Windows, this will first search the SDK Store (as above) and after that, the Shared Credentials File. On Linux, only the Shared Credentials File will be searched. If a profile is found, the code will return the credentials for that profile. If a profile could not be found in the chain, an exception is thrown.

7: ENVIRONMENT VARIABLES

The third delegate that is added attempts to create an instance of EnvironmentVariablesAWSCredentials which derives from the AWSCredentials class. The constructor for this type calls a FetchCredentials method which searches for configured environment variables.

```
1   public ImmutableCredentials FetchCredentials()
2   {
3       string accessKeyId = Environment.GetEnvironmentVariable(ENVIRONMENT_VARIABLE_ACCESSKE
4       string secretKey = Environment.GetEnvironmentVariable(ENVIRONMENT_VARIABLE_SECRETKEY)
5       if (string.IsNullOrEmpty(secretKey))
6       {
7           secretKey = Environment.GetEnvironmentVariable(LEGACY_ENVIRONMENT_VARIABLE_SECRET
8           if (!string.IsNullOrEmpty(secretKey))
9               logger.InfoFormat("AWS secret key found using legacy and non-standard environ
10                                LEGACY_ENVIRONMENT_VARIABLE_SECRETKEY, ENVIRONMENT_VARIABLE_
11      }
12
13      if (string.IsNullOrEmpty(accessKeyId) || string.IsNullOrEmpty(secretKey))
14      {
15          throw new InvalidOperationException(string.Format(CultureInfo.InvariantCulture,
16              "The environment variables {0}/{1}/{2} were not set with AWS credentials.",
17              ENVIRONMENT_VARIABLE_ACCESSKEY, ENVIRONMENT_VARIABLE_SECRETKEY, ENVIRONMENT_VA
18      }
19
20      string sessionToken = Environment.GetEnvironmentVariable(ENVIRONMENT_VARIABLE_SESSION_
21
22      logger.InfoFormat("Credentials found using environment variables.");
23
24      return new ImmutableCredentials(accessKeyId, secretKey, sessionToken);
25  }
```

view raw (https://gist.github.com/stevejgordon/e6dd8e57af4bc1e01cdc169220c4cbf2/raw/a75fe918b554dabd4b78326b3b8404ce563bcda1/EnvironmentVariablesAWSCredentials.cs) EnvironmentVariablesAWSCredentials.cs (https://gist.github.com/stevejgordon/e6dd8e57af4bc1e01cdc169220c4cbf2#file-environmentvariablesawscredentials-cs) hosted with ❤ by **GitHub (https://github.com)**

The access key is expected to be stored in an environment variable “AWS_ACCESS_KEY_ID”. The secret access key is expected in the environment variable “AWS_SECRET_ACCESS_KEY” or the legacy “AWS_SECRET_KEY” environment variable. This code also looks for an “AWS_SESSION_TOKEN” environment variable which may be set if you are using temporary credentials. This can be the case if you use the AWS Security Token Service to provide short-lived credentials. AWS uses the session token to validate the temporary security credentials.

At a minimum, the access key ID and the secret key must be located. An instance of ImmutableCredentials is then created and returned.

8: ECS TASK CREDENTIALS OR EC2 INSTANCE CREDENTIALS

The last generator attempts to load credentials from locations which may be available if you have deployed your service on AWS using either ECS (Elastic Container Service) or an EC2 instance. When running your services in production, a best practice is not to manually provide credentials but to rely on IAM roles which can be assigned to EC2 instances and ECS tasks. This allows AWS to manage the credentials for the instance or task by providing credentials that are granted access permissions from an IAM role. This gets into some deeper security territory about how these features work which I will gloss over here.

The code which loads the ECS/EC2 credentials is as follows:

```
1   private static AWSCredentials ECSEC2CredentialsWrapper(IWebProxy proxy)
2   {
3       try
4       {
5           string uri = System.Environment.GetEnvironmentVariable(ECSTaskCredentials.Contain
6           if (!string.IsNullOrEmpty(uri))
7           {
8               return new ECSTaskCredentials(proxy);
9           }
10      }
11      catch (SecurityException e)
12      {
13          Logger.GetLogger(typeof(ECSTaskCredentials)).Error(e, "Failed to access environme
14      }
15      return DefaultInstanceProfileAWSCredentials.Instance;
16  }
```

view raw (https://gist.github.com/stevejgordon/620a8fec3fae95c58b1087ccca684159/raw/6af8bb2a0bad90b57980003484da66868a24e5f2/FallbackCredentialsFactory.cs) FallbackCredentialsFactory.cs (https://gist.github.com/stevejgordon/620a8fec3fae95c58b1087ccca684159#file-fallbackcredentialsfactory-cs) hosted with ❤ by **GitHub (https://github.com)**

In short, when your service is running as a container on ECS, and a task role is applied, the Amazon ECS agent populates an environment variable "AWS_CONTAINER_CREDENTIALS_RELATIVE_URI" for all containers that belong to the task with a relative URI. The code above checks to see if this environment variable is set with a relative URI and if so uses a URIBasedRefreshingCredentialHelper to load the credentials.

When running directly on an EC2 instance, the instance role will be used to fetch credentials from the ECS instance metadata. The DefaultInstanceProfileAWSCredentials is used to access a cached

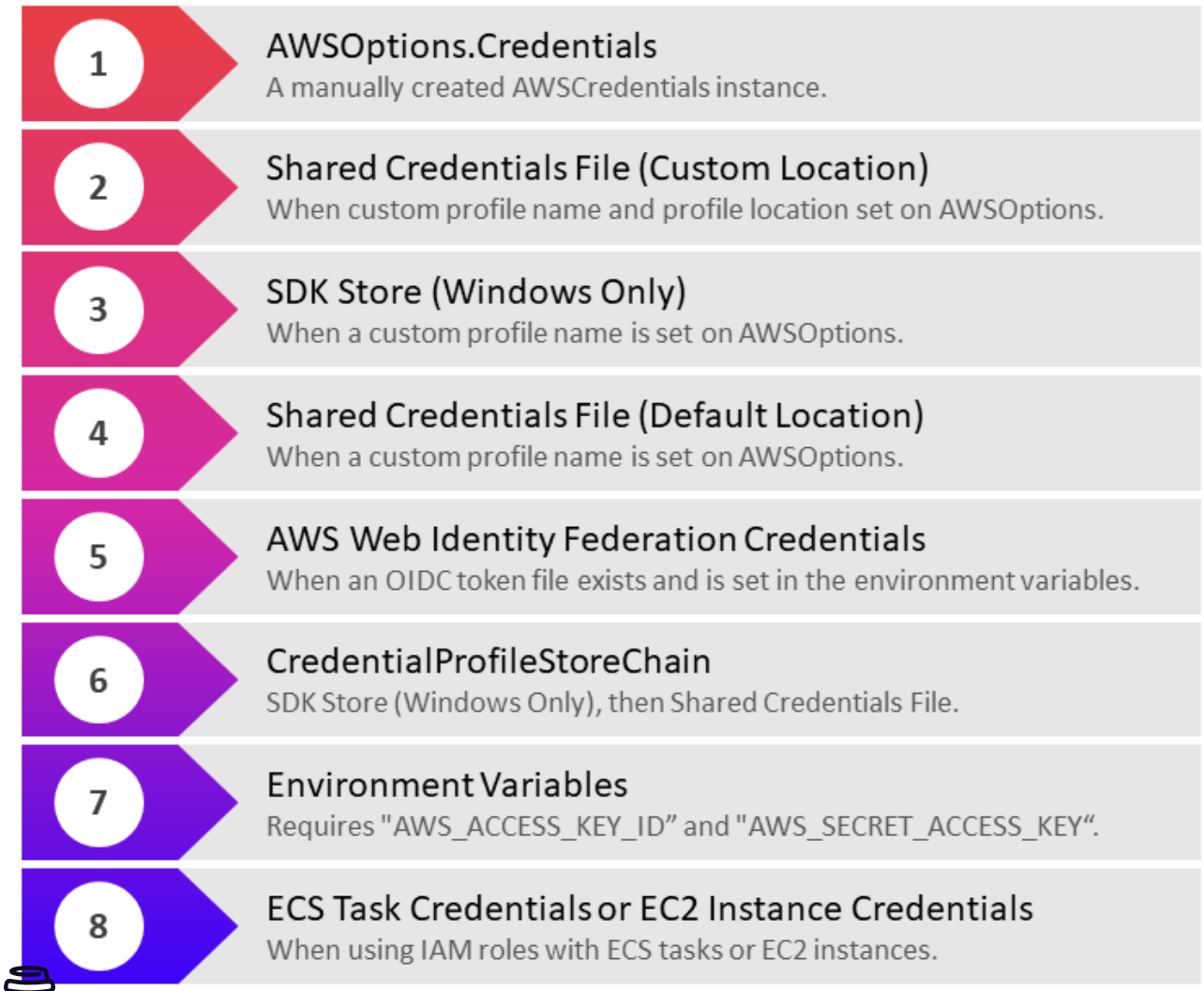
instance of the credentials which refreshes every two minutes based on the EC2 instance metadata.

IAM roles for EC2 instances and ECS tasks are the recommended way to supply credentials. In both of these cases, you should not need to load the credentials manually. Instead, allow the SDK to load them for you. This will occur automatically unless you have provided credentials using any of the methods which are checked first.

SUMMARY

That's way more information than you probably needed. If you've reached this far, well done indeed! While the internal implementation for credential loading is not something you need to know to this depth, I find this useful background knowledge to understand the sources which may supply credentials. If your service is failing to load credentials or is using credentials which do not grant the expected access, understanding how these resolve can be useful.

Here are the steps, one more time.



Have you enjoyed this post and found it useful? If so, please consider supporting me:

Buy me a coffee(https://www.buymeacoffee.com/stevejgordon)

(https://www.paypal.com/cgi-bin/webscr?cmd=_s-xclick&hosted_button_id=WV4JPPV9FS34L&source=url)

aws sdk (https://www.stevejgordon.co.uk/tag/aws-sdk)

credentials (https://www.stevejgordon.co.uk/tag/credentials) Sdk (https://www.stevejgordon.co.uk/tag/sdk)

STEVE GORDON (HTTPS://WWW.STEVEJGORDON.CO.UK/AUTHOR/STEVEJGORDON)

Steve Gordon is a Microsoft MVP, Pluralsight author, senior engineer and community lead. He works for Elastic (https://www.elastic.co). maintaining their .NET language clients. Steve is passionate about community and all things .NET related, having worked with ASP.NET for over 16 years. Steve enjoys sharing his knowledge through his blog, in videos and by presenting at user groups and conferences. Steve is excited to be a part of the .NET community and founded .NET South East, a .NET Meetup group based in Brighton. He enjoys contributing to and maintaining OSS projects. You can find Steve on Twitter as @stevejgordon (https://twitter.com/stevejgordon)

ALSO ON **STEVE GORDON'S BLOG**

Lexcess • 2 years ago

Nice little article, this stuff always feels harder than it should be. Quick pro tip you can replace: C:\Users\stevejgordon\.aws\credentials with %userprofile%\.aws\credentials in most cases to make it generic.

1 • Reply • Share ›

Esteban Villegas • a year ago

Thank you very much for this post, it had the answer to what I was looking for for over a day.

• Reply • Share ›

Hagatorn • a year ago

I think I'm following your advice as I should be.

I've tried loading from a named profile both Craig and default. I've also tried adding environmental variables. However, unless I manually create a Credential object it doesn't authenticate and the credentials object remains blank.

```
services.AddSingleton<iamazondynamodb>(sp =>
{


var clientConfig = root.GetAWSOptions();
var credentials = new BasicAWSCredentials(
System.Environment.GetEnvironmentVariable("AWS_ACCESS_KEY_ID"),
System.Environment.GetEnvironmentVariable("AWS_SECRET_ACCESS_KEY ")
);
return new AmazonDynamoDBClient(credentials, clientConfig.Region );
});
```

• Reply • Share ›

Steve Gordon Mod ➦ Hagatorn • a year ago

Hi,

Cheers,
Steve

1 ^ | v • Reply • Share ›

Hagatorn ➚ Steve Gordon • a year ago

Thanks, the posted code is a workaround as I failed to get your suggestion to work. I just opened a StackOverflow Q: https://stackoverflow.com/q...

^ | v • Reply • Share ›

Hagatorn ➚ Steve Gordon • a year ago • edited

Amazing to get such a quick response. So I started out following one of your other posts and my original code was:

```
services.AddAWSService<iamazondynamodb>();
```

Then I added the following above.

```
services.AddDefaultAWSOptions(root.GetAWSOptions());
```

Despite having a Profile: "x" and the environmental variables for the keys the settings don't seem to be getting pulled through.
I actually made an issue:
https://github.com/aws/aws-...

^ | v • Reply • Share ›

Maycon Fernando Silva Brito • a year ago

Thank you Steve. Your article is awesome! It's clearly and better than the AWS documentation.

^ | v • Reply • Share ›

Oluwaseun Akinrinlola • a year ago

please how do i use the no 7 option.
I am using docker to run my web api application so i have a .env file. How do i read these values from the .env file in the Startup.cs ConfigureServices method

^ | v • Reply • Share ›

Kevin Rich • 2 years ago

Hey, Steve, this is a great primer for an article that is not easy to find in the AWS docs. I'd love to see a followup showing your recommended development flow when setting up your DI. I'm a fan of having the SDK Credential store configured locally and roles configured on the deployment target resource. What do you use?

^ | v • Reply • Share ›

Steve Gordon **Mod** ➚ Kevin Rich • 2 years ago

Hi Kevin,

I'm glad this filled a gap. Credential Store for dev + roles for production is a good pattern. For our day-to-day development, it's a tad more complex. We have access keys in a credential file which provide very limited access to AWS. We then use a script to assume an appropriate role in the environment we need to work in. Those assumed

• Reply • Share ›

Alexsandro • 2 years ago

Work with AWS Credential is too bad😫. Nice post to try make it more easy.

• Reply • Share ›

‹ Do We Have an Obsession with Ducks in Software Development? (https://www.stevejgordon.co.uk/do-we-have-an-obsession-with-ducks-in-software-development)

Architecting a Cloud-Native Service with .NET and AWS › (https://www.stevejgordon.co.uk/architecting-a-cloud-native-service-with-dotnet-and-aws)

(https://mvp.microsoft.com/en-us/PublicProfile/5002866)

PLURALSIGHT COURSES

(https://pluralsight.pxf.io/c/1394957/1194718/7490)

Dependency Injection in ASP.NET Core (https://pluralsight.pxf.io/4KQD9)

Using Configuration and Options in .NET Core and ASP.NET Core Apps (https://pluralsight.pxf.io/Ny152)

Building ASP.NET Core Hosted Services and .NET Core Worker Services (https://pluralsight.pxf.io/vdn6j)

Integration Testing ASP.NET Core Applications: Best Practices (https://pluralsight.pxf.io/m7rX1)

Implementing Cross-cutting Concerns for ASP.NET Core Microservices (https://pluralsight.pxf.io/xeXXy)

String Manipulation in C#: Best Practices (https://pluralsight.pxf.io/EaEV1P)

FOLLOW ME

(htt
ps://
ww
w.yo
utu
be.c
om
/cha
nnel
/UC_
rMp
ypH
GP8
_J8A
Ao_
bLC
mA?
&yt
bCh
ann
el=n
ull)

(htt
ps://
twitt
er.c
om
/ste
vejg
ordo
n)

(htt
ps://
gith
ub.c
om
/ste
vejg
ordo
n)

m
/in/s
teve
-
gord
on-
a669
1114
/)

Subscribe

Signup for news and updates direct to your inbox.

Email

You can unsubscribe anytime. For more details, review our Privacy Policy.

Opt in to receive news and updates.

SUPPORT THIS BLOG

Become a patron

Buy me a coffee!(https://www.buymeacoffee.com/stevejgordon)

Search...

RECENT POSTS

Playing with System.Text.Json Source Generators (https://www.stevejgordon.co.uk/playing-with-system-text-json-source-generators)

How Does the StringBuilder Work in .NET? (Part 3) (https://www.stevejgordon.co.uk/how-does-the-stringbuilder-work-in-net-part-3-how-appending-works-and-the-stringbuilder-expands)

How Does the StringBuilder Work in .NET? (Part 2) (https://www.stevejgordon.co.uk/how-does-the-stringbuilder-work-in-dotnet-part-2-understanding-the-overhead)

How Does the StringBuilder Work in .NET? (Part 1) (https://www.stevejgordon.co.uk/how-does-the-stringbuilder-work-in-dotnet-part-1-why-do-we-need-a-stringbuilder-and-when-should-we-use-one)

Using DateOnly and TimeOnly in .NET 6 (https://www.stevejgordon.co.uk/using-dateonly-and-timeonly-in-dotnet-6)

YouTube 1K

.NET SOUTH EAST

(https://www.meetup.com /dotnetsoutheast/)

Please come and join our new .NET User Group in Brighton, UK.

CATEGORIES

Select Category

ARCHIVES

Select Month

(htt
ps://
ww
w.yo
utu
be.c
om
/cha
nnel
/UC_
rMp
ypH
GP8
_J8A
Ao_
bLC
mA?
&yt
bCh
ann
el=n
ull)

(htt
ps://
twitt
er.c
om
/ste
vejg
ordo
n)

(htt
ps://
gith
ub.c
om
/ste
vejg
ordo
n)

(htt
ps://
www.linkedin.com/in/steve-gordon-a6691114/)

TWITTER (HTTPS://TWITTER.COM/STEVEJGORDON) GITHUB (HTTPS://GITHUB.COM/STEVEJGORDON)

LINKEDIN (HTTPS://WWW.LINKEDIN.COM/IN/STEVE-GORDON-A6691114/)

YOUTUBE (HTTPS://WWW.YOUTUBE.COM/CHANNEL/UC_RMPYPHGP8_J8AAO_BLCMA?&YTBCHANNEL=NULL)